UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de la Telecomunicación

TRABAJO FIN DE GRADO

Aprendizaje continuo mediante redes convolucionales

Anselmo Velázquez Pazos

Tutor: Álvaro García Martín

Ponente: José María Sánchez Martínez

Enero 2021

Aprendizaje continuo mediante redes convolucionales

AUTOR: Anselmo Velázquez Pazos
TUTOR: Álvaro García Martín

Video Processing and Understanding Lab
Dpto. 111 Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero 2021

Trabajo parcialmente financiado por el Gobierno de España bajo el proyecto TEC2017-88169-R (MobiNetVideo)

Resumen

La principal atención de este trabajo de fin de grado es el estudio y utilización de algoritmos o técnicas de aprendizaje continuado para diferentes conjuntos de datos e imágenes.

En primer lugar, se hará una breve introducción a los conceptos básicos de la Inteligencia Artificial, a las redes neuronales convolucionales y a su aplicación en el mundo del Aprendizaje Profundo, dado que comprender y conocer este tipo de redes es importante para el aprendizaje continuado.

En segundo lugar, se describirán los conjuntos de datos utilizados y sus correspondientes atributos dado que se cuenta con un conjunto de imágenes propias proporcionadas por el Video Processing and Understanding Lab (VPU) de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid (UAM). Además, se hará una breve introducción al entorno de desarrollo que se ha utilizado para la realización de este trabajo y a los algoritmos utilizados en los diferentes experimentos.

En tercer lugar, se explicarán los criterios de evaluación de los resultados obtenidos, así como la evolución de los mismos durante la utilización de diferentes algoritmos de aprendizaje continuado. Se realizarán diferentes experimentos en los cuales se extraerán conclusiones y resultados gracias a los cuales se verá la importancia y la potencia del aprendizaje continuado o aprendizaje continuo.

Finalmente se proporcionarán pruebas visuales que demuestren los resultados obtenidos y se propondrán vías de investigación que podrían llevarse a cabo en el futuro.

Palabras Clave

Aprendizaje profundo, aprendizaje continuado, precisión, dominio, clase, tarea, olvido catastrófico.

Abstract

The main focus of this final degree project is the study and use of algorithms or continuous learning techniques for different sets of data and images.

First, there will be a brief introduction to the basic concepts of Artificial Intelligence, convolutional neural networks and their application in the world of Deep Learning, since understanding and knowing this type of network is important for continual learning.

Second, the datasets used and their given attributes are described. There is a set of own images provided by the Video Processing and Understanding Lab (VPU) of the UAM Escuela Politécnica Superior. In addition, a brief introduction will be made to the development enviroment that has been used to carry out this work and to the algorithms used in the different experiments.

Third, the evaluation criteria of the results obtained will be explained, as well as their evolution during the use of different algorithms for continual learning.

Different experiments will be carried out in which conclusions and results will be drawn, thanks to which the importance and power of continual learning will be seen.

Finally, visual tests will be provided to demonstrate the results obtained and avenues of investigation that could be carried out in the future will be proposed.

Keywords

Deep Learning, continual learning, accuracy, domain, class, task, catastrophic forgetting.

Agradecimientos

Agradezco a mis padres, en primer lugar, por haber estado siempre ahí y por haberme apoyado en cada etapa académica de mi vida.

También agradezco a mis abuelos, Pepe y Amalia, por haberme ayudado durante toda la etapa del colegio y a mi abuela Luisa por haberme cuidado siempre que ha hecho falta.

Quiero dar las gracias a mi novia Ana por haberme apoyado y acompañado en los últimos años de carrera.

Por último, quiero dar las gracias a mi tutor Álvaro por todo lo que me ha ayudado.

Indice

Indice de ilustraciones

Indice de tablas

.

1. INTRODUCCIÓN

En este capítulo se introducirán las razones por las que se realiza este TFG. Esta sección se divide en:

- Motivación
- Objetivos
- Organización de la memoria

1.1 Motivación

El Aprendizaje Profundo (*Deep Learning)* es una de las áreas más importantes e interesantes dentro del mundo de la Inteligencia Artificial. Actualmente se sabe que dentro del Aprendizaje Profundo se pueden construir redes neuronales muy útiles y potentes como las redes convolucionales.

Sin embargo, el Aprendizaje Profundo tiene algún que otro punto débil. Aunque las redes neuronales artificiales están inspiradas en la biología, los diseños y los métodos de aprendizaje de las mismas difieren sustancialmente de las redes neuronales biológicas. Los parámetros de las redes neuronales artificiales son aprendidos a través de un conjunto de datos y después son utilizados estáticamente sobre nuevos datos. Para acomodarse a los diferentes cambios en lo que se refiere a la distribución de los datos, las redes neuronales artificiales tienen que ser re-entrenadas con el nuevo conjunto de datos para evitar *overfitting* y olvidos catastróficos.

Para mitigar estos olvidos catastróficos se podría pensar que bastaría con entrenar desde cero con los antiguos y con los nuevos datos. Sin embargo, esto resultaría inviable dado que hay redes que tardan días en ser entrenadas, por lo que esta solución rudimentaria supondría un elevadísimo coste computacional y de tiempo.

Teniendo en cuenta que la capacidad de aprender de los seres humanos se realiza de forma incremental y que por ello el cerebro humano no sufre olvidos catastróficos, en este trabajo se trata de aplicar diferentes algoritmos y técnicas que permitan mitigar estos olvidos catastróficos en redes neuronales artificiales ([1] Yen-Chang Hsu, 2018).

1.2. Objetivos

El objetivo de este trabajo de fin de grado es la utilización y comparación de diferentes algoritmos y técnicas que permiten introducir el Aprendizaje Continuado o Incremental en las redes neuronales convolucionales, lo que provocaría una reducción o mitigación de los olvidos catastróficos.

Pasos para lograr este objetivo:

- Se hará una introducción y estudio previo de los elementos fundamentales del *Deep Learning* y de las redes convolucionales, lo que dará una visión general del tema tratado en este trabajo.
- Se procesará y se adaptará el conjunto de imágenes que utilizaremos en los algoritmos.
- Se determinarán los diferentes parámetros y atributos a utilizar en los algoritmos tratados en este trabajo.
- Una vez elegidos los parámetros, ejecutaremos los algoritmos para el conjunto de datos utilizado en este trabajo.
- Tras la ejecución, se analizarán los resultados.
- Finalmente se aportarán pruebas visuales tanto de la ejecución como de los resultados obtenidos.

1.3. Organización de la memoria

La memoria contiene los siguientes capítulos:

- **Capítulo 1: Introducción.** Descripción de los objetivos y estructura del trabajo.
- **Capítulo 2*:* Estado del arte.** Desarrollo del estado del arte y de los conceptos necesarios para desarrollar el trabajo.
- **Capítulo 3*:* Diseño y desarrollo**. Presentación de los conjuntos de datos, técnicas y librerías utilizadas, así como el desarrollo del trabajo.
- **Capítulo 4*:* Análisis de los algoritmos** utilizados e interpretación de los resultados obtenidos.
- **Capítulo 5*:* Conclusiones y trabajo futuro**

2. Estado del arte

2.1. Introducción

Para la correcta comprensión de este trabajo es necesario ponerse en situación y entender lo que es el Aprendizaje Automático (*Machine Learning*). El Aprendizaje Automático es uno de los campos de la Inteligencia Artificial, se puede definir como la ciencia que permite que los ordenadores aprendan y actúen de forma similar a los humanos, es decir, mejorando su aprendizaje a lo largo del tiempo de forma completamente autónoma siendo alimentados con datos en forma de observaciones o interacciones con el mundo real. Se puede decir que el Aprendizaje Automático ofrece una manera altamente eficiente de capturar el conocimiento mediante la información que contienen los datos para mejorar gradualmente el rendimiento de los modelos de predicción. El Aprendizaje Automático es una tecnología que actualmente se encuentra presente en múltiples situaciones: reconocimiento de voz, reconocimiento de imágenes (que es en lo que se centra este trabajo), conducción autónoma.

Dentro del mundo de la Inteligencia Artificial existe un campo denominado Aprendizaje Profundo (*Deep Learning*), la idea principal detrás de este concepto es tratar de parecerse al cerebro humano para trata de replicar o reproducir su comportamiento informáticamente. El Aprendizaje Profundo nace como una solución al tratamiento de datos masivos dado que los algoritmos de este campo obtienen como resultado patrones o características absolutamente indetectables por el ser humano. Dentro del Aprendizaje Profundo se puede ubicar a las redes neuronales, que son la base fundamental de todo. En este trabajo se va a hablar sobre algoritmos de Aprendizaje Continuado aplicados a un determinado tipo de red neuronal: las redes neuronales convolucionales. Las redes neuronales convolucionales se utilizan en diversas aplicaciones relacionadas con el reconocimiento de imágenes.

2.2. Redes Neuronales

El punto de partida del estudio de las redes neuronales artificiales podría fijarse en pleno siglo XX con las investigaciones de Santiago Ramón y Cajal dado que fue él quien desarrolló la idea de la neurona como componente más pequeño y básico en la estructura cerebral.

En la mayoría de estudios e investigaciones sobre redes neuronales artificiales se realizan analogías entre las neuronas biológicas y las puertas de silicio (componentes básicos de los ordenadores). En términos de velocidad se puede decir

que las neuronas biológicas son hasta seis veces más lentas. Sin embargo, el cerebro cuenta con un número de interconexiones muy grande. También hay que añadir que el cerebro humano es mucho más eficiente desde un punto de vista energético.

La capacidad del cerebro humano para realizar operaciones en paralelo permite realizar tareas que para un ordenador son realmente complicadas ya que requieren una elevadísima cantidad de cálculos. Se puede poner el ejemplo de un reconocimiento de un determinado animal en una fotografía, aunque la foto esté mal hecha o con el animal girado el cerebro no tiene ningún problema en ejecutar el reconocimiento mientras que este giro supone un serio problema para un ordenador.

Se puede definir a una red neuronal artificial como un modelo matemático desarrollado para emular el cerebro humano.

Con todo esto se puede decir que las redes neuronales artificiales y el cerebro humano tienen en común la existencia de una serie de elementos básicos: las neuronas.

Las neuronas artificiales están interconectadas entre sí mediante unas conexiones denominadas pesos sinápticos. Estos pesos van cambiando a lo largo del tiempo gracias al proceso de aprendizaje ([2] Lancomilla, 2016).

El esquema general de una neurona presenta el siguiente aspecto:

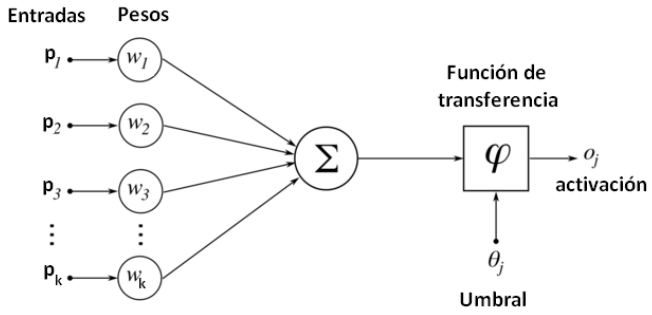

Ilustración 2.1: Esquema general de una neurona
*Fuente: **([21] Caparrini, 2019)***

2.2.1. Entrenamiento

Para poder hablar del entrenamiento de las redes neuronales hay que dejar claros varios conceptos antes. En primer lugar, hay que comprender las diversas partes del esquema de la neurona artificial (elemento fundamental y básico de toda red neuronal):

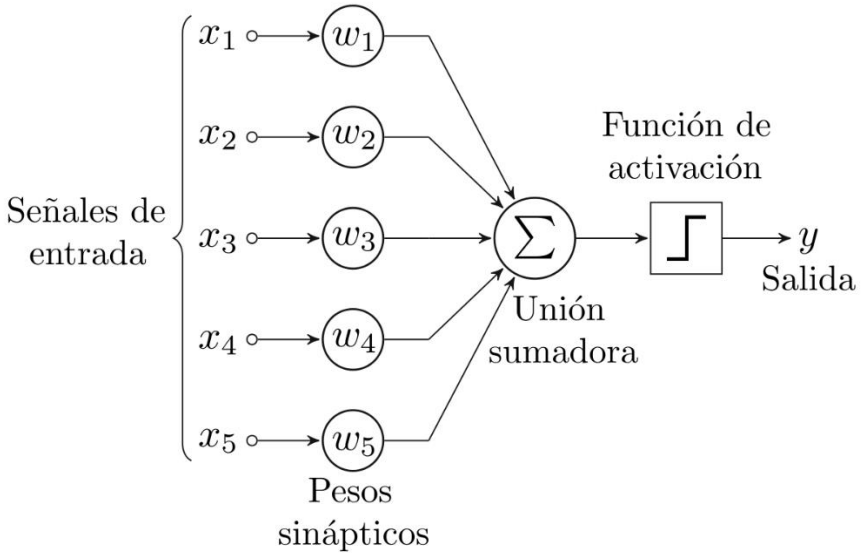

Ilustración 2.2: Esquema analítico de una neurona
*Fuente: **([22] Cancela, 2017)***

Con el entrenamiento lo que se pretende es que las redes neuronales aprendan de forma autónoma qué parámetros deben utilizar una vez conocidos los datos de entrada. Es importante saber que toda red neuronal tiene una estructura parecida a la siguiente:

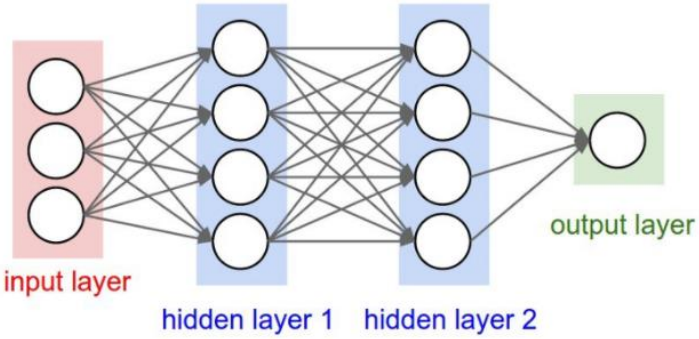

Ilustración 2.3: Esquema genera de una red neuronal
*Fuente: **([23] Kane, 2017)***

El proceso de entrenamiento de una red neuronal podría definirse de una forma muy sencilla:

- Algoritmo de optimización: el más utilizado es el descenso por gradiente.
- Algoritmo de propagación hacia atrás (*backpropagation*).

El resumen que se puede hacer del entrenamiento de las redes neuronales podría ser la siguiente: se usa el algoritmo de propagación hacia atrás para calcular el vector gradiente que será utilizado en el algoritmo de descenso por gradiente para llevar a cabo la optimización de la función de coste.

El concepto de función de coste es muy importante. Sin duda es muy necesario conocer bien este concepto a la hora del diseño de redes neuronales dado que es la función que define el error para cada combinación de parámetros de la red. Cuanto menor sea el valor de la función de coste, mejor. Para entenderlo mejor se puede decir que la función de coste mide la "diferencia" entre el valor de salida y el valor actual. De esta forma, la red neuronal se va alimentando en diferentes iteraciones con los mismos datos de entrenamiento y en cada iteración se intenta minimizar la función de coste. Esta minimización se consigue ajustando los pesos de las conexiones entre neuronas al final de cada iteración. La función de coste más utiliza es el error cuadrático medio (MSE), con la siguiente fórmula:

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}_{\text{The square of the difference between actual and predicted}}^{2}$$

Ilustración 2.4: Fórmula matemática MSE

*Fuente: **([24] Pascual, 2018)***

Otro concepto realmente importante es el de tasa de aprendizaje. Se puede definir a la tasa de aprendizaje como el hiperparámetro que determina la velocidad a la que aprende una red neuronal. Cuanto menor es la tasa de aprendizaje más tiempo se tardará en minimizar la función de coste. Sin embargo, a medida que la tasa de aprendizaje es más grande aumenta la posibilidad de no minimizar de forma óptima la función de coste. Pueden darse los siguientes casos:

- Tasa de aprendizaje demasiado pequeña:

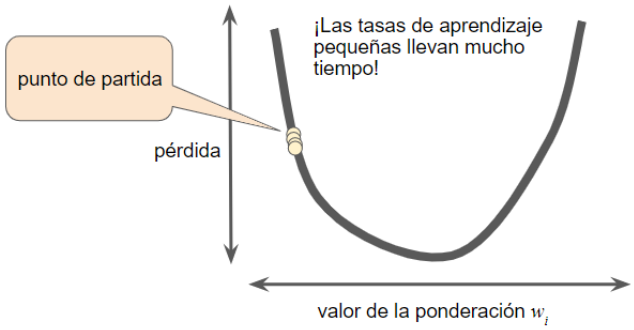

Ilustración 2.5: Gráfica tasa de aprendizaje demasiado pequeña
*Fuente: **([25] Developers Google, s.f.)***

- Tasa de aprendizaje demasiado grande:

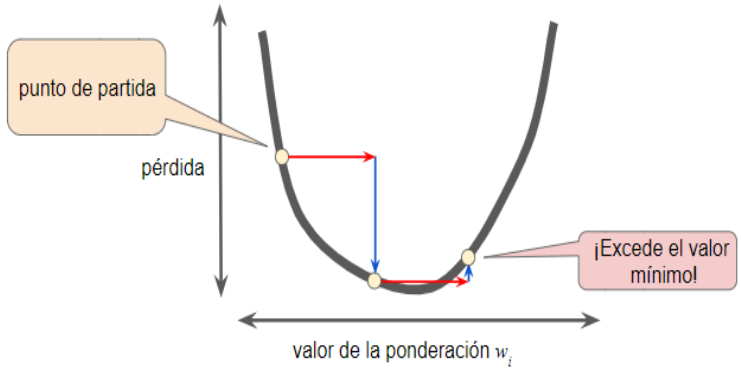

Ilustración 2.6: Gráfica tasa de aprendizaje demasiado grande
*Fuente: **([25] Developers Google, s.f.)***

- Tasa de aprendizaje correcta:

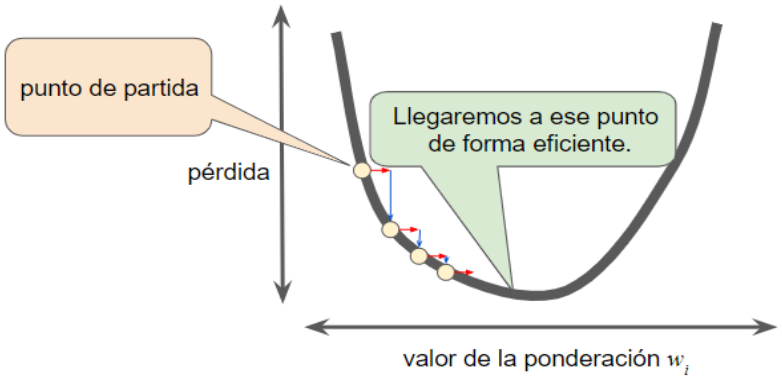

Ilustración 2.7: Gráfica tasa de aprendizaje correcta
*Fuente **([25] Developers Google, s.f.)***

2.2.2. Otros conceptos importantes

En esta sección del trabajo se va a hablar sobre otros conceptos de redes neuronales totalmente necesarios para comprender el funcionamiento de las mismas.

En primer lugar, vamos a hablar de los ciclos (*epoch*). El número de ciclos de una red neuronal es el número de veces que se ejecutarán los algoritmos de *forwardpropagation* y *backpropagation*. En cada ciclo TODOS los datos de entrenamiento pasar por la red neuronal para que esta aprenda. Por ejemplo, si la idea de red neuronal consiste en 20 ciclos y 1000 datos, por la red neuronal pasa cada ciclo con los 1000 datos. El número de ciclos es un hiperparámetro.

En segundo lugar, se menciona el tamaño de batch (*batchsize*). Si se especifica este parámetro, se puede decir que cada ciclo va a tener más ejecuciones a nivel interno, a cada una de estas ejecuciones se le puede denominar iteración. Por lo tanto, el tamaño de batch es un concepto que se puede definir como el número de datos que tiene cada iteración de un ciclo.

Otro concepto importante es el del aumento de datos (*data augmentation*). El aumento de datos consiste en aplicar alteraciones sobre los datos originales para producir diferentes tipos de modificaciones en ellos y así aumentar el número de datos. Con este aumento del número de datos lo que se busca es mejorar los resultados dado que al entrenar la red con un número más grande de datos lo que se consigue es una mayor robustez de la misma. Las alteraciones más utilizadas son rotaciones, cambios en el brillo, giros horizontales o verticales…

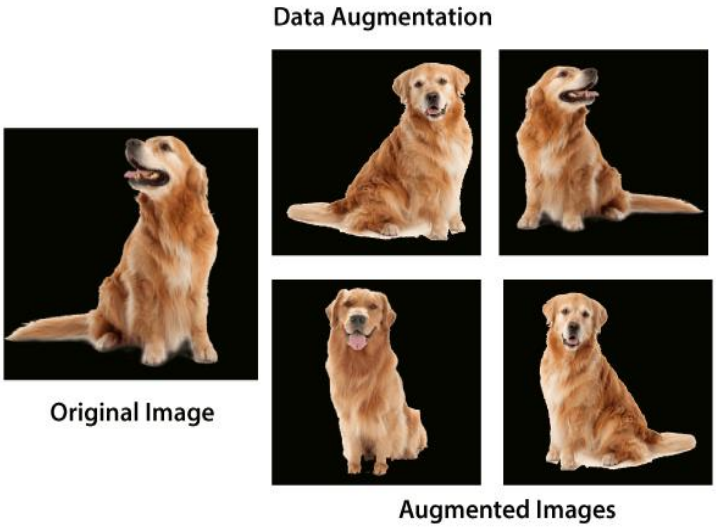

Ilustración 2.8: Ejemplo de Data Augmentation
*Fuente: **([26] Gondhalekar, 2020)***

2.3. Redes Neuronales Convolucionales (CNN)

En este trabajo se utiliza un determinado tipo de redes neuronales: las redes neuronales convolucionales (*CNN*). Las redes neuronales convolucionales son utilizados en multitud de aplicaciones, pero sobre todo en el reconocimiento, análisis y clasificación de imágenes. Las redes neuronales convolucionales trabajan de una forma muy parecida a las redes neuronales convencionales con la diferencia de que en el caso de las redes convolucionales cada neurona es un filtro bidimensional (incluso tridimensional) que está convolucionado con la entrada de la capa, esto es verdaderamente importante a la hora de tratar con el aprendizaje de patrones como los de una imagen ([3] Bagnato, 2018).

Una red neuronal convolucional suele presentar un aspecto similar al de la siguiente figura:

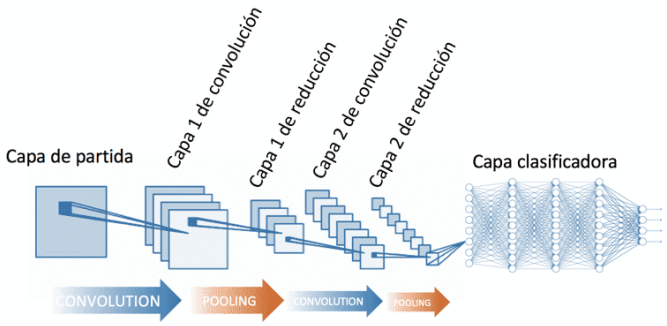

Ilustración 2.9: Esquema general de una red neuronal convolucional
*Fuente: **([27] López González, 2017)***

Como se puede apreciar en la anterior imagen, una red neuronal convolucional presenta diversos tipos de capas. Las capas de convolución o convolucionales son las más relevantes. Estas capas básicamente son un conjunto de filtros que son convolucionados con una determinada entrada dando lugar a un mapa de características en una salida. En otras palabras, se puede decir que las redes neuronales toman como entrada una imagen y van aplicando los filtros a las diferentes partes de dicha imagen obteniendo como resultado algo mucho más fácil y óptimo de procesar. Es muy común denominar a los filtros como *kernels*. Una vez finaliza este "filtrado" se aplica una función de activación, la más utilzada es la llamada *ReLu*, que lo que hace es cambiar los valores negativos por el valor "0". Con esta última fase lo que se busca es incrementar la no-linealidad de la imagen.

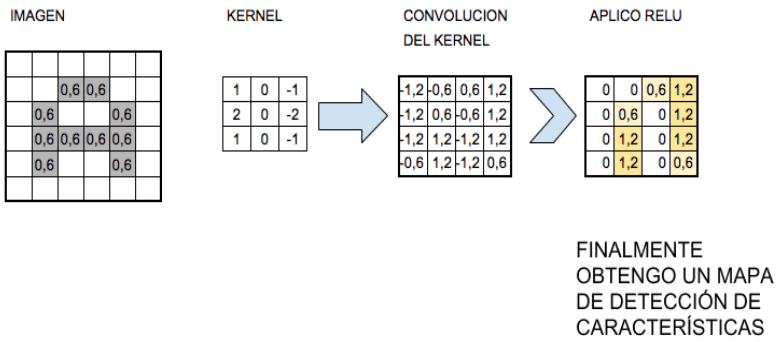

Ilustración 2.10: Ejemplo de aplicación del kernel y ReLu
*Fuente: **([28] Bagnato, 2018)***

Existen dos formas de ajustar el tamaño de la imagen de salida de este tipo de capa ([4] Chollet, 2017):

- Relleno (*Padding*): consiste básicamente en añadir ceros a los bordes de la imagen. Con el relleno lo que se consigue es que tras aplicar un *kernel* 3x3 a una imagen 6x6 se obtiene una salida 6x6, es decir, se preserva la dimensión de entrada:

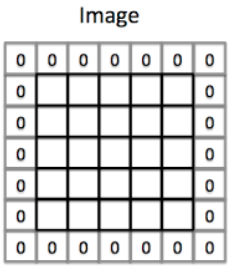

Ilustración 2.11: Ejemplo de padding
*Fuente: **([28] Bagnato, 2018)***

- Paso (*Stride*): Es el parámetro que define el salto del *kernel* cuando se aplica sobre la imagen, es decir, lo que avanza cada que se lleva a cabo una iteración. Si se define un paso elevado se analiza menos información y se disminuye el tiempo de computación mientras que si se define un paso pequeño, se analiza más información obteniendo una imagen de salida grande, aunque aumenta significativamente el tiempo de procesamiento.

El otro tipo de capa que aparece en las redes neuronales convolucionales es denominada como capa *pooling*. La función de esta capa es muy sencilla: reducir el tamaño de la imagen de salida de la capa de convolución. De esta forma se reduce el coste computacional. El objetivo del *pooling* es quedarse con las características más importantes. Existen dos formas de hacerlo:

- *Max-pooling:* consiste en dividir la imagen en regiones de las cuales se extrae el píxel de mayor valor.

- *Average-pooling:* consiste en dividir la imagen en regiones, de cada región se calcula una media del valor de sus píxeles y se devuelve ese valor medio.

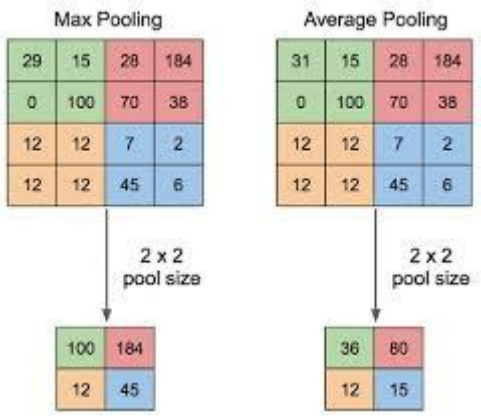

Ilustración 2.12: Ejemplo de pooling
Fuente: ***([29] Muhamad, 2019)***

Una vez finalizada la fase de diversas convoluciones y *pooling*, se pasa a una fase clasificadora (primera figura). Es decir, la salida se conecta a la denominada capa *fully connected* que tiene la característica fundamental de que todas sus neuronas están interconectas con las de la capa anterior y posterior. Finalmente se aplica una última capa que transforma los valores de salida de la capa *fully connected* en valores probabilísticos que pueden ser utilizados en la tarea de clasificiación. Lo más utilizado es *softmax*, que lo que hace básicamente es asignar una probabilidad a cada clase en función de los valores recibidos. La función *softmax* presenta la siguiente expresión:

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{j} e^{y_j}}$$

Ilustración 2.13: Fórmula matemática de la función SOFTMAX
Fuente: ***([30] Numerntur.org)***

3. Diseño y desarrollo

3.1. Introducción

En este capítulo se expondrá de forma explicativa y de forma gráfica los diferentes conjuntos de datos utilizados en este trabajo: MNIST y CONTENEDORES BENIDORM. También se explicará el funcionamiento de la librería Pytorch y el porqué de su utilización. También se habla sobre el *Transfer Learning* dado que también ha sido utilizado en gran medida en este trabajo.

En este capítulo se llevará a cabo la explicación del concepto de Aprendizaje Continuo. Este punto tiene una importancia bastante relevante dado que el trabajo gira en torno a este concepto. Una vez explicado el concepto de Aprendizaje Continuo se procede a explicar las redes neuronales empleadas.

Por último, se explica el pre-procesamiento realizado para poder utilizar las imágenes del conjunto de datos CONTENEDORES BENIDORM, así como el desarrollo del trabajo y sus diferentes fases.

3.2. Datasets

3.2.1. MNIST

La base de datos MNIST es un conjunto de imágenes de dígitos escritos a mano. La base de datos consta de diversas imágenes de los dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

Esta base de datos posee un conjunto de entrenamiento de 60.000 ejemplos y un conjunto de prueba o *test* de 10.000 ejemplos. Las imágenes que constituyen esta base de datos se obtuvieron gracias a una normalización en tamaño de las imágenes originales en blanco y negro para caber en un cuadro de 20x20 píxeles mientras se conserva su relación de aspecto. Las imágenes resultantes de esta normalización contienen niveles de gris debido a la técnica anti-aliasing utilizada por el algoritmo de normalización.

Es una base de datos diseñada por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos de América. Esta base de datos se suele utilizar para la capacitación de diversos sistemas de procesamiento de imágenes.

Dentro de la base de datos MNIST se puede hablar de varios subconjuntos: SD-1 y SD-3. SD-1 contiene 58.527 imágenes de dígitos escritos por 500 escritores diferentes. La diferencia entre SD-1 y SD-3 es que en SD-3 los bloques de datos de cada escritor aparecen en secuencia mientras que en SD-1 aparecen codificados. El

conjunto de entrenamiento MNIST está compuesto por 30.000 patrones SD-3 y 30.000 patrones SD-1. El conjunto de prueba está constituido por 5.000 patrones SD-3 y 5.000 patrones SD-1 ([5] LeCun Yann, s.f.).

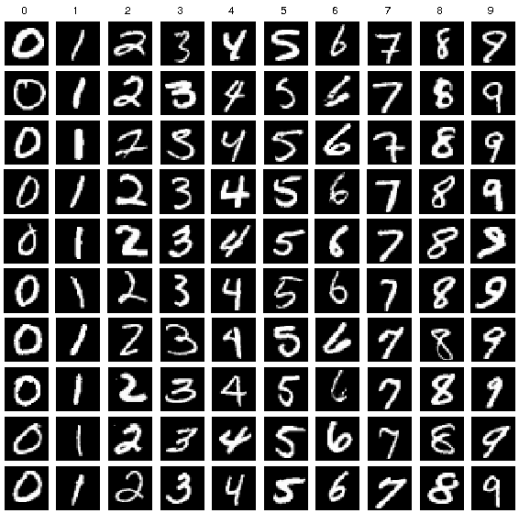

Ilustración 3.1: Ejemplo del conjunto de datos MNIST
*Fuente: **([31] Lim, 2016)***

3.2.2 Contenedores Benidorm

Este es el conjunto de datos más relevante para este trabajo. El conjunto de datos consiste básicamente en imágenes de Benidorm, muchas de ellas contienen contenedores y otras muchas no contienen contenedores. En este trabajo se considera por contenedor a un recipiente dentro del cual se introducen diferentes tipos de residuos. Dentro de las que contienen contenedores hay siete tipos de contenedores diferentes (aceite, orgánica, vidrio, papel, envases, ropa y resto). El conjunto de datos consta en total de 1.095 imágenes con contenedores y 17.726 imágenes sin contenedores. Se ha organizado el conjunto de entrenamiento de tal forma que cuenta con el 85% de las imágenes de ambos tipos y el conjunto de prueba cuenta con el 15% restante. Se ha seguido los estándares de distribución para el conjunto de entrenamiento y prueba utilizados para MNIST.

Para poder utilizar este conjunto de datos en este trabajo se ha tenido que llevar a cabo una tarea de pre-procesamiento en Matlab en la cual, para las imágenes con contenedores se ha llevado a cabo la selección del contenedor de cada imagen recortándolo del resto de la imagen (gracias a un fichero de coordenadas de recorte proporcionado por el VPULab) para posteriormente redimensionarlo para obtener una imagen final de 32x32 píxeles. Esta búsqueda de una imagen final de 32x32 píxeles se debe a que el tipo de red neuronal utilizada funciona correctamente con estos tamaños de imagen. Para las imágenes sin contenedor lo que se ha hecho ha sido utilizar unas coordenadas de recorte para recortar diferentes partes de cada

una de las imágenes (todo ello en Matlab). Estas coordenadas de recorte se obtuvieron calculando los valores medios de las coordenadas utilizadas para recortar los contenedores. De esta forma se obtiene un archivo con el valor medio de las cuatro coordenadas que se necesitan para efectuar un recorte. Este archivo de coordenadas de recorte puede asegurar que no se han seleccionado zonas poco coherentes en relación a la posición natural ocupada por los contenedores en las imágenes que sí que contienen dichos contenedores (un ejemplo de algo a evitar es seleccionar un trozo de cielo azul dado que no tendría ningún sentido a la hora del entrenamiento), después de esto también se redimensionaron las imágenes para que tuvieran un tamaño de 32x32 píxeles ([6] Krizhevsky, 2009). En resumen: lo que se busca para las imágenes sin contenedor es recortar un trozo de cada una de ellas para poder llevar a cabo todo el pre-procesamiento de manera similar al proceso realizado para las imágenes con contenedor, además se busca que estos trozos seleccionados tengan sentido (seleccionar un trozo de cielo azul no tiene ningún sentido).

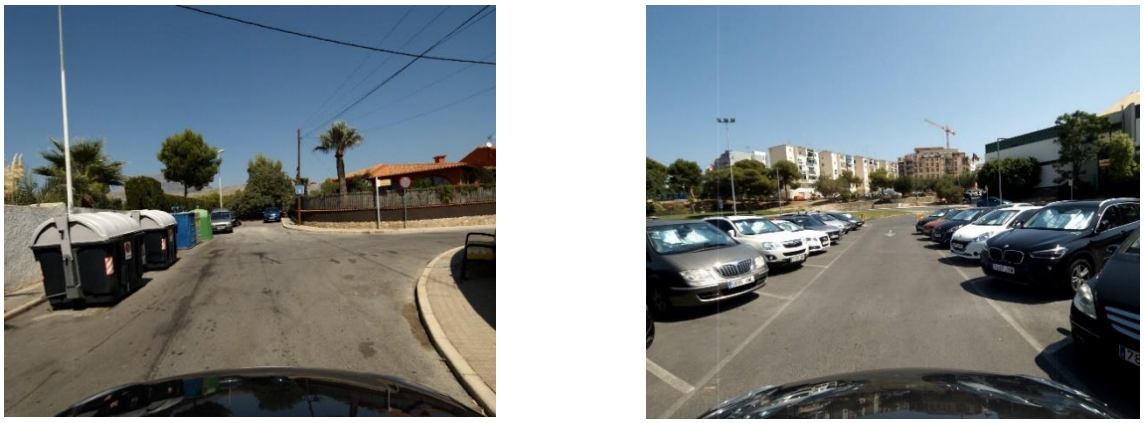

Ilustración 3.2: Ejemplos del conjunto de datos CONTENEDORES BENIDORM

Las dos imágenes anteriores son un ejemplo de los dos tipos: CON CONTENEDOR y SIN CONTENEDOR.

En resumen, se puede decir que hay 8 clases:

1. NO CONTENEDOR
2. ACEITE
3. VIDRIO
4. ROPA
5. PAPEL
6. ORGÁNICO
7. ENVASES
8. RESTO

3.3. Técnicas y librerías

3.3.1. Pytorch

En este trabajo se ha utilizado la librería Pytorch en gran medida. Es una librería diseñada y desarrollada por Facebook en el año 2017. Se está hablando de una librería de código abierto basado en Python.

Gracias a una API de NVIDIA denominada CUDA que consigue conectar la CPU con la GPU, Pytorch es de lo más utilizado en el mundo del *Deep Learning* dado que permite la utilización de tensores, la utilización de la GPU para realizar los distintos tipos de cálculos e incluso la utilización de la tarjeta gráfica para estas funciones. Esta capacidad de utilización de la GPU hace que Pytorch consiga reducir en gran medida los tiempos de computación gracias a la paralelización de procesos ([7] Tran, 2020).

3.3.2. Transfer Learning

El *transfer learning* o transferencia del aprendizaje es una técnica muy utilizada dentro del mundo del *Deep Learning* y en este trabajo ha sido utilizada en cierta forma. Esta técnica consiste básicamente en utilizar redes neuronales que han sido entrenadas con anterioridad para una determinada tarea. Antes de utilizar estas redes se llevan a cabo una serie de modificaciones de cara a realizar una tarea similar con otros datos.

La ventaja del *transfer learning* es principalmente el ahorro de tiempo dado que entrenar una red desde cero es algo que puede conllevar mucho tiempo ([8] Dipanjan (DJ), 2018) ([9] Brownlee, 2017).

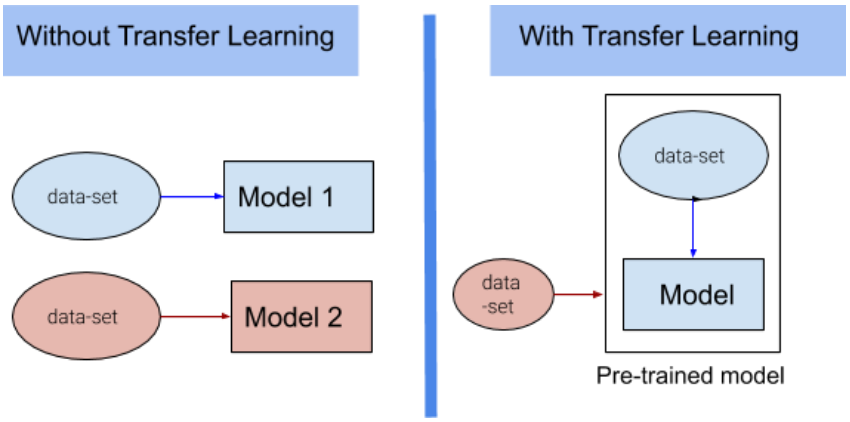

Ilustración 3.3: Ejemplo Transfer Learning
*Fuente: **([32] Hussain, 2020)***

3.4. Aprendizaje Continuo

Se va a hablar del Aprendizaje Continuo en el contexto de tareas de reconocimiento de imágenes dado que es de lo que trata en líneas generales este trabajo. La definición general del concepto de Aprendizaje Continuo podría ser: capacidad de aprender algo nuevo aprovechando lo aprendido anteriormente.

Para comprender mejor el concepto de Aprendizaje Continuo resulta muy útil comprender un par de ejemplos.

Un ejemplo muy bueno para comprender este concepto es el del diseño de un dron cuyo objetivo es detectar desde el cielo números de serie de antenas, la corrosión presente en cada antena y el tipo de antena. A continuación, se entrena al dron usando redes neuronales artificiales para que reconozca todo lo anterior, pero, ¿qué pasa si con el paso del tiempo se da otro tipo de situación para la cual el dron no ha sido entrenado? Por ejemplo, ahora se quiere que el dron vuele en otro país donde las antenas son totalmente diferentes. Por lo tanto, lo que se necesita ahora es entrenar la red neuronal en torno a estas nuevas clases.

Otro buen ejemplo es el de los coches autónomos. Se afronta ahora otro caso de detección de objetos y segmentación de imágenes. Se da el caso de que nuestro método o sistema no funciona bien para un determinado caso (una esquina o algo parecido). Lo que se haría sería etiquetar estos casos y entrenar la red neuronal con estos nuevos datos etiquetados. Se puede pensar que únicamente tendríamos que entrenar a la red neuronal artificial con los nuevos datos. Es decir, tendríamos una Tarea A qué consistiría en entrenar de forma general a la red neuronal y una Tarea B que consistiría en entrenar a la red con los nuevos datos etiquetados del fallo comentado. Lo que suele ocurrir es que obtiene un buen rendimiento para la Tarea B y un rendimiento algo peor para la Tarea A, esto se debe a que los nuevos conocimientos interfieren en los antiguos, es lo que se denomina olvido catastrófico.

Cuando se habla sobre Aprendizaje Continuo es totalmente necesario comprender dos conceptos:

1. **Dilema PLASTICIDAD-ESTABILIDAD:**

   El desarrollo de algoritmos de Aprendizaje Continuo trata de balancear plasticidad vs estabilidad.

   La dificultad del Aprendizaje Continuo viene de este dilema:

   - Plasticidad: se necesita una red neuronal con buena plasticidad en el sentido de que sea posible que la red aprenda cosas nuevas de forma rápida y eficaz.
   - Estabilidad: se necesita una red neuronal estable que no olvide cosas antiguas.

2. **Regularización:**

   En este trabajo sólo se habla sobre aproximaciones que utilizan REGULARIZACIÓN. En líneas generales la regularización consiste en actualizar los pesos de la red y llevar a cabo penalizaciones a los cambios con el objetivo de minimizar al máximo los olvidos.

El principal problema que los algoritmos de Aprendizaje Continuo deben resolver es el olvido catastrófico. El olvido catastrófico representa el gran problema del *Deep Learning* ya que sin algoritmos de Aprendizaje Continuo la única forma de enseñar nuevas tareas es entrenar la red que se tenga desde cero cada vez que se quiera aprender una nueva tarea ([10] Aljundi, 2019)

Se puede dividir los algoritmos de Aprendizaje Continuo de la siguiente manera ([11] Maltoni & Lomonaco, 2018):

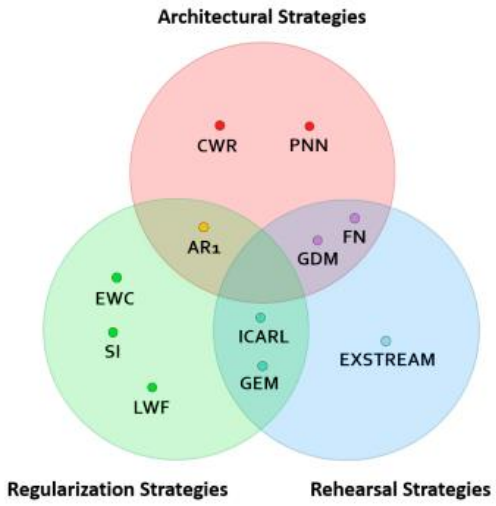

Ilustración 3.4: Esquema áreas del Aprendizaje Continuo
*Fuente: **([33] Lomonaco, 2020)***

En este trabajo se tratan únicamente algunas *Regularization Strategies*.

Formas de mitigar los olvidos catastróficos:

- Como se ha dicho antes, entrenando desde cero con los nuevos y con los antiguos datos. Esto genera múltiples problemas: en muchos casos las redes neuronales tardan días o semanas en ser entrenadas, se da un tremendo gasto de energía y computación, puede que los datos antiguos no sean accesibles…
- La otra forma sería utilizar algoritmos de Aprendizaje Continuo.

3.5. Redes Neuronales empleadas

3.5.1. Resnet

La red neuronal convolucional utilizada para el conjunto de datos CONTENEDORES BENIDORM (que es lo que compete principalmente a este trabajo) es la ResNet o *Residual Network.*

ResNet es una red neuronal convolucional creada por Microsoft en el año 2015 y su función consiste básicamente en introducir un aprendizaje residual.

ResNet está formada por bloques residuales. Estos bloques añaden una conexión de salto de capas, lo que permite a la entrada sumarse o incluirse a la salida de cada una de estas capas de forma que la red lo que hace es aprender de la diferencia entre ellas. Con todo esto ResNet consigue poner fin al problema del desvanecimiento de gradiente que se produce cuando se aumenta el número de capas de una red neuronal. Se puede decir que ResNet fue la primera red neuronal convolucional que consiguió aumentar el número de capas sin perder precisión.

En este trabajo se ha utilizado una Wide ResNet denominada *WideResNet_28_2_cifar*. Las Wide ResNet son una novedosa arquitectura derivada de las clásicas ResNet en las que se disminuye la profundidad y se aumenta el ancho de las redes residuales con el objetivo de poner fin al problema (presente en las ResNet) de la disminución de la reutilización de características, lo que hace que las ResNet fueran muy lentas a la hora de entrenar.

A continuación, se puede apreciar de forma gráfica un ejemplo de un esquema general característico de una ResNet:

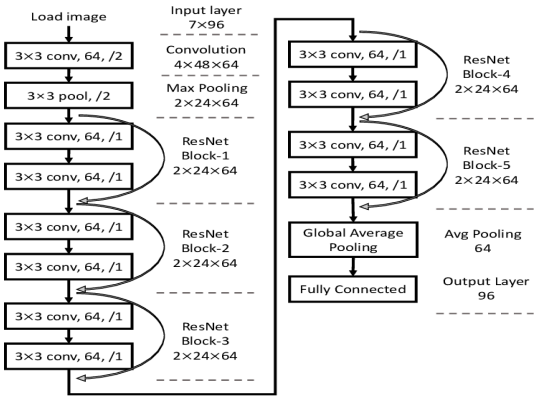

Ilustración 3.5: Esquema general de ResNet
*Fuente: **([34] Hyungeun, Seunghyoung, & Hongseok, 2018)***

3.6. Pre-Procesamiento

Para poder utilizar el conjunto de datos de Contenedores Benidorm se ha tenido que llevar a cabo un pre-procesamiento importante.

Para las imágenes con contenedores se ha utilizado un fichero que contiene las coordenadas de ubicación del contendor/contenedores dentro de cada imagen. El pre-procesamiento de estas imágenes ha consistido en recortar el contenedor de cada una de ellas utilizando esas coordenadas. Una vez recortado cada contenedor se ha llevado a cabo una transformación de cada una de las imágenes para que tengan un tamaño de 32x32.

Para las imágenes sin contenedores lo que se ha hecho ha sido calcular un valor medio para cada una de las cuatro coordenadas utilizadas para el recorte de contenedor en las imágenes con contenedor de forma que se obtienen cuatro valores medios con los cuales se efectúan los recortes para cada una de las imágenes sin contenedores. Esto se ha hecho para evitar realizar recortes en partes de las imágenes que no tienen sentido como sería el cielo azul o fachadas de edificios. Una vez realizados los recortes se han realizado las transformaciones pertinentes para obtener un tamaño de 32x32 en cada una de las imágenes recortadas.

Una vez hecho todo lo anterior, se llevó a cabo una división de las imágenes en conjunto de entrenamiento y de test. Se asignó el 85% de las imágenes con contenedor y el 85% de las imágenes sin contenedor al conjunto de entrenamiento y el 15% restante de cada una al conjunto de test.

En el pre-procesamiento también se llevó a cabo la creación de un archivo csv en el que figura el nombre de cada imagen seguido por su respectiva etiqueta o tipo de contenedor/no contenedor.

3.7. Desarrollo

Una vez finalizado el pre-procesamiento de las imágenes se procede al inicio del desarrollo.

Lo primero que se hace es cargar las imágenes de entrenamiento y de validación. Para poder llevar a cabo esta carga fue necesario la creación de un script específico para el conjunto de datos CONTENEDORES BENIDORM dado que, por defecto, el código está diseñado para ser utilizado con el conjunto de datos MNIST. Este script termina devolviendo la imagen y la etiqueta de la misma en el formato requerido.

Tras finalizar el proceso de carga del conjunto de datos en el proyecto se lleva a cabo un proceso de *SPLIT* en el que se divide cada conjunto (entrenamiento y validación) en diferentes tareas en función de sus clases:

split_boundaries: [0, 2, 4, 6, 8]

{'1': [0, 1], '2': [2, 3], '3': [4, 5], '4': [6, 7]}

Se puede apreciar cómo se ha organizado el conjunto de datos de forma que la tarea 1 consiste en reconocer si la imagen introducida en el proyecto pertenece a la clase 0 (no contenedor) o a la clase 1 (imagen con un determinado tipo de contenedor). En este proyecto también se ha tenido en cuenta el siguiente caso de *SPLIT*:

split_boundaries: [0, 1, 2, 3, 4, 5, 6, 7, 8]

{'1': (0, 1), '2': (0, 2), '3': (0, 3), '4': (0, 4), '5': (0, 5), '6': (0, 6), '7': (0, 7)}

Se puede apreciar como en una cada de las tareas hay dos clases: la primera clase siempre es un 0 (que representa una imagen sin contenedor) y la segunda clase siempre es un número entre el 1 y el 7 (que representan una imagen con un determinado tipo de contenedor).

Una vez comprendido el pre-procesamiento llevado a cabo, así como el proceso de *SPLIT*, se puede proceder a introducir y analizar los diferentes escenarios posibles. La siguiente imagen resulta útil a la hora de plasmar los distintos escenarios:

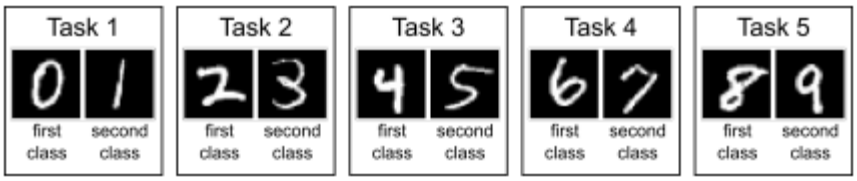

Ilustración 3.6: Ejemplo de split para MNIST

*Fuente: **([35] M. van de Ven, 2019)***

La imagen anterior sirve para cómo sería el *SPLIT* en el caso del conjunto de datos MNIST. En el caso del conjunto de datos CONTENEDORES BENIDORM, en lugar de imágenes con números; en cada imagen figura una imagen con un tipo de contenedor o, en el caso de la clase 0, figura una imagen sin contenedor:

- {'1': [0, 1], '2': [2, 3], '3': [4, 5], '4': [6, 7]} → En este caso, la *Task 1* está compuesta de imágenes sin contenedor (*first class*) e imágenes con contenedor de tipo 1 (*second class*).
- {'1': (0, 1), '2': (0, 2), '3': (0, 3), '4': (0, 4), '5': (0, 5), '6': (0, 6), '7': (0, 7)} → En el otro caso tratado en este trabajo cada *Task* está compuesta por imágenes sin contenedor (*first class*) e imágenes de un tipo de contenedor (*second class*).

A continuación, se explica en qué consiste cada escenario ([12] Gido M. van de Ven1, 2019):

- *Incremental Task:* Con la tarea dada o conocida, ¿es la primera o la segunda clase? En MNIST, ¿0 ó 1? En CONTENEDORES BENIDORM, ¿no contenedor ó contenedor tipo 1?
- *Incremental Domain:* Con la tarea desconocida, ¿es la primera o la segunda clase? En MNIST, ¿[0,2,4,6,8] ó [1,3,5,7,9]? En CONTENEDORES BENIDORM, ¿[0,2,4,6] ó [1,3,5,7]?
- *Incremental Class:* Con la tarea desconocida… En MNIST, ¿qué dígito es? En CONTENEDORES BENIDORM, ¿qué tipo de contenedor es?

Seguidamente se explica el proceso fundamental que se lleva cabo. Lo primero que se hace es llamar desde la consola al ejecutable, por ejemplo: *split_DATA_CONTENEDORES_incremental_task.sh.* A continuación, se va a analizar secuencialmente el desarrollo del trabajo:

1. En primer lugar, se deben elegir los parámetros (en el ejecutable sh) que queremos de:
    - *Batch size.*
    - *Regularization coefficient.*
    - *Optimizer.*
    - *Split size.*
    - *Schedule*

2. En segundo lugar, aparece una representación de cada uno de los parámetros introducidos en la red neuronal convolucional que se va a utilizar en la hora del entrenamiento. En este trabajo se utiliza siempre una red denominada *WideResNet_28_2_cifar* que, como el nombre indica, es de tipo *Wide ResNet*:

```
PreActResNet_cifar(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (stage1): Sequential(
    (0): PreActBlock(
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

```
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (shortcut): Sequential(
     (0): Conv2d(16, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
   )
   (1): PreActBlock(
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
   )
   (2): PreActBlock(
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
   )
   (3): PreActBlock(
    (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
   )
 )
```

Tal y como se puede apreciar, aparece por pantalla toda la información en relación a la red: *padding, stride, kernel size…*

3. En tercer lugar, se podrá apreciar como comienza la ejecución del algoritmo seleccionado. Dada la gran cantidad de información que aparecerá por pantalla, lo más útil es esperar a que finalice la ejecución ya que se podrá apreciar el siguiente resumen del proceso:

```
===Summary of experiment repeats: 10 / 10 ===
```

The regularization coefficient: 100.0
The last avg acc of all repeats: [11.44067797 8.89830508 7.20338983 21.39830508
19.27966102 8.68644068
20.76271186 12.71186441 18.00847458 22.66949153]
mean: 15.10593220338983 std: 5.621437874314486
reg_coef: 100.0 mean: 15.10593220338983 std: 5.621437874314486

Básicamente este resumen coloca de forma muy intuitiva el resumen de cada repetición del experimento (en este caso se había elegido el parámetro 10, de forma que se ejecutan diez repeticiones o pasadas). De esta forma por pantalla se puede apreciar el coeficiente de regularización utilizado en el experimento, la media de precisión obtenida en cada una de las repeticiones, la media de precisión general (la media de todas las medias) de todo el experimento y la desviación típica general de todo el experimento. Cambiando diferentes parámetros como el coeficiente de regularización se puede llevar a cabo un ajuste de la desviación típica que se ve reflejado en la media general también. Es importante decir que la media de precisión de cada una de las repeticiones representa la media de aciertos (a la hora de devolver la clase de una imagen), es decir, una media de 11.44067797 es una media bastante baja (cada 100 se producen unos 11 aciertos). En el último capítulo de este trabajo se lleva a cabo una interpretación en detalle de estos resultados.

4. Evaluación

4.1. Introducción

En este capítulo se realizará un análisis de los diferentes algoritmos utilizados en este trabajo, así como un análisis de las métricas utilizadas para evaluar el rendimiento y el funcionamiento de cada uno de los algoritmos de Aprendizaje Continuado tratados en este trabajo. También se procederá a analizar los resultados obtenidos tras la utilización de diferentes parámetros y se procederá a una comparación de los resultados obtenidos para los conjuntos de datos MNIST y CONTENEDORES BENIDORM.

4.2. Marco de evaluación

El Aprendizaje Continuo se puede considerar como un paso adelante en el mundo del *Deep Learning* totalmente necesario dado que el problema de los olvidos catastróficos es algo que debe ser eliminado. Para lograr acercarnos a este Aprendizaje Continuo en este trabajo se han tratado diversos algoritmos: EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017), SI ([19] Zenke, Poole, & Ganguli, 2017) y MAS ([20] Rahaf Aljundi, 2018).Los resultados serán comparados con los resultados obtenidos de otros algoritmos que no utilizan Aprendizaje Continuo (*baselines*): SGD ([13] Roy, 2000), ADAM ([14] Kingma & Lei Ba, 2017), ADAGRAD ([15] Ruder, 2016) y L2 ([16] Durán, 2019).

4.2.1. Algoritmos

4.2.1.1. ALGORITMOS BASELINES

4.2.1.1.1. SGD

Antes de explicar el método Descenso por Gradiente Estocástico (SGD ([13] Roy, 2000)) es necesario comprender correctamente lo que es el Descenso por Gradiente típico.

El Descenso por Gradiente es una técnica de optimización muy utilizada en *Machine Learning* y *Deep Learning* que puede ser aplicada en la mayoría de los algoritmos de aprendizaje. Un gradiente es la pendiente de una función, mide el grado de cambio de una variable en respuesta a los cambios de otra variable. Matemáticamente, el Descenso por Gradiente es una función convexa cuya salida es la derivada parcial de

un conjunto de parámetros de sus entradas. Cuanto mayor sea el gradiente, más pronunciada será la pendiente. Empezando desde un valor inicial, el Descenso por Gradiente es ejecutado iterativamente con el objetivo de buscar los valores óptimos de los parámetros para encontrar el mínimo valor posible para la función de coste.

Hay tres tipos de Descenso por Gradiente:

1. *Batch Gradient Descent.*
2. *Stochastic Gradient Descent.*
3. *Mini-batch Gradient Descent.*

A continuación, se explica en más detalle el método SGD ([13] Roy, 2000). La palabra "estocástico" significa "sistema o proceso que está conectado a una probabilidad aleatoria". En SGD ([13] Roy, 2000) algunas muestras se seleccionan aleatoriamente en lugar del conjunto de datos completo para cada iteración. En Descenso por Gradiente hay un término denominado *batch* el cual hace referencia al número total de muestras de un conjunto de datos que son utilizadas para calcular el gradiente en cada iteración.

En *Batch Gradient Descent* el *batch* seleccionado es el conjunto de datos completo. Usar el *dataset* completo es muy útil para obtener el mínimo de una forma menos ruidosa y menos aleatoria; el problema viene cuando el conjunto de datos es muy grande. Si el conjunto de datos es de dos millones de muestras, se necesitarían esos dos millones de muestras para completar una simple iteración, lo cual supone un coste computacional muy elevado.

El problema anterior es resuelto por SGD ([13] Roy, 2000) dado que usa de forma general una muestra por *batch*. En SGD ([13] Roy, 2000) dado que sólo se elige al azar una muestra del conjunto de datos para cada iteración, la ruta que toma el algoritmo para alcanzar los mínimos suele ser más ruidosa que en el caso de Descenso por Gradiente. Sin embargo, esto no es realmente relevante dado que no importa el camino que tome el algoritmo siempre que se alcance el mínimo y con un tiempo de entrenamiento más corto.

En las siguientes imágenes se pueden apreciar las diferentes rutas que siguen los algoritmos:

- Ruta seguida por *Batch Gradient Descent:*

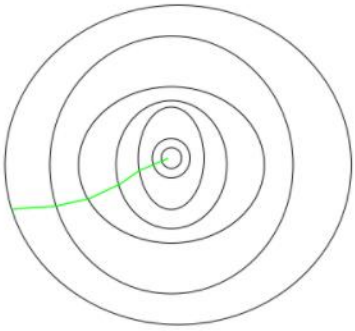

Ilustración 4.1: Ruta método Batch Gradient Descent
*Fuente: **([36] Rahul, 2020)***

- Ruta seguida por SGD ([13] Roy, 2000):

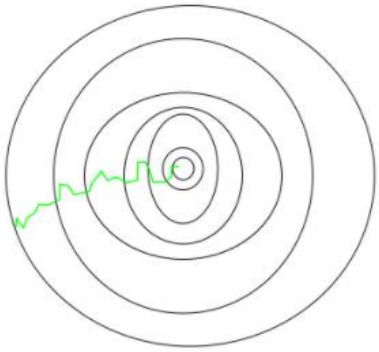

Ilustración 4.2: Ruta método SGD
*Fuente: **([36] Rahul, 2020)***

Como se puede apreciar en las imágenes anteriores, SGD ([13] Roy, 2000) es más ruidoso que el *Batch Grandient Descent* dado que toma un mayor número de iteraciones para alcanzar los mínimos. A pesar de ello, sigue siendo menos costoso a nivel computacional. Es por ello que en la mayoría de escenarios se prefiere a SGD.

4.2.1.1.2 ADAM

ADAM ([14] Kingma & Lei Ba, 2017) es un algoritmo para la optimización basada en gradientes de primer orden de funciones objetivas estocásticas, las cuales están basadas en estimaciones adaptativas de momentos de pequeño orden o *low-order moments*. Este método es sencillo de implementar, es eficiente a nivel computacional, tiene pocos requerimientos de memoria y es, en cierta forma, adecuado para problemas que son grandes en términos de datos o parámetros. Este método también es apto para objetivos no estacionarios y problemas con gradientes

muy ruidosos y/o escasos. Los hiperparámetros que se introducen en este método tienen como ventajas que presentan interpretaciones intuitivas y que no requieren grandes ajustes. Diferentes resultados empíricos demuestran que el método Adam funciona bien en la práctica y en comparación a otros métodos de optimización estadística.

La optimización basada en el gradiente estocástico es algo muy importante y muy utilizado en el mundo de la Inteligencia Artificial y de la ingeniería en general. Muchos problemas dentro de la Inteligencia Artificial se pueden presentar como la optimización de alguna función objetivo escalar parametrizada que requiere maximización o minimización con respecto a sus parámetros.

Este método lo que hace es calcular las tasas de aprendizaje adaptativo individuales para diferentes estimaciones de parámetros de primer y segundo orden de los gradientes.

El nombre ADAM ([14] Kingma & Lei Ba, 2017) procede de "estimación del momento adaptativo" (siglas en inglés). Este método combina las ventajas de otros dos métodos muy conocidos: AdaGrad y RMSProp.

Ventajas: Algunas de las ventajas de ADAM ([14] Kingma & Lei Ba, 2017) son que las magnitudes de las actualizaciones de parámetros no varían en relación al cambio de escala del gradiente, sus tamaños de paso o *stepsizes* están delimitados de forma aproximada por el hiperparámetro de tamaño de paso o *stepsize* y el método trabaja con gradientes escasos.

A continuación, se puede observar el rendimiento de ADAM ([14] Kingma & Lei Ba, 2017) en el ejemplo de regresión logística para el conjunto de datos MNIST:

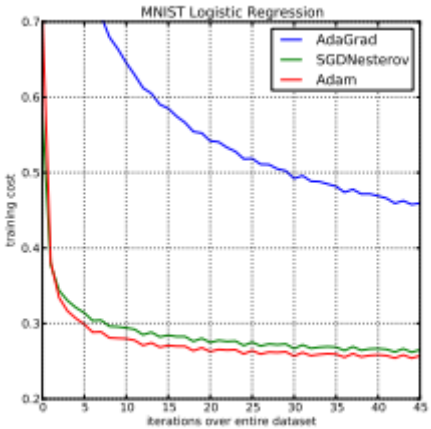

Ilustración 4.3: Gráfico rendimiento ADAM

*Fuente: **([37] An, Yu, & Changjian, s.f.)***

4.2.1.1.2 ADAGRAD

ADAGRAD ([15] Ruder, 2016) es un algoritmo para la optimización basada en gradientes. ADAGRAD ([15] Ruder, 2016) realiza lo siguiente: adapta la tasa de aprendizaje a los parámetros realizando actualizaciones pequeñas para los parámetros asociados que poseen características que se dan con frecuencia, y actualizaciones más grandes para los parámetros asociados con características poco frecuentes. Por esta razón, ADAGRAD ([15] Ruder, 2016) es adecuado para tratar con datos escasos.

Diferentes estudios establecen que ADAGRAD ([15] Ruder, 2016) mejora enormemente la solidez de SGD ([13] Roy, 2000). A raíz de esto, ADAGRAD ([15] Ruder, 2016) se utilizó para entrenar redes neuronales a gran escala en Google. También se ha utilizado ADAGRAD ([15] Ruder, 2016) para entrenar las incrustaciones de palabras GloVe, dado que las palabras poco frecuentes requieren actualizaciones mucho más grandes que las frecuentes.

Se puede resumir todo lo anterior diciendo que el principio fundamental del funcionamiento de este algoritmo es el siguiente: ADAGRAD ([15] Ruder, 2016) usa una tasa de aprendizaje diferente para cada parámetro θ en cada paso de tiempo t.

4.2.1.1.3 L2:

El objetivo de este tipo de método es reducir el valor de los parámetros para que sean pequeños. Este método introduce un término novedoso y adicional: la penalización. Esta penalización se introduce en la función de coste original (**L**): la suma del cuadrado de los parámetros (**w**).

La parte negativa de la incorporación de este es que puede ser alto, tanto que la red minimizaría tanto la función de coste que haría los parámetros muy cercanos a cero, lo que no sería óptimo. Debido a ello se multiplica el sumando por una constante (**λ**), cuyo valor se escoge de forma arbitraria: 0.1, 0.001…

La función de coste en L2 ([16] Durán, 2019) queda así:

$$L_2(X,\omega) = L(X,\omega) + \lambda \sum \omega_i^2$$

Ilustración 4.4: Fórmula matemática función de coste en L2
*Fuente: **([16] Durán, 2019)***

Si se quisiera aplicar más regularización simplemente habría que aumentar el valor de **λ.**

4.2.1.2. ALGORITMOS DE APRENDIZAJE CONTINUO:

4.2.1.2.1. EWC:

En el cerebro la consolidación sináptica permite el aprendizaje continuo al reducir la plasticidad de las sinapsis que son fundamentales para las tareas aprendidas con anterioridad. Este algoritmo, denominado *Elastic Weight Consolidation*, realiza una operación similar en redes neuronales artificiales a la llevada a cabo en el cerebro humano dado que restringe parámetros importantes para que se mantengan cerca de sus valores anteriores. En este algoritmo se trata de encontrar una solución para una nueva tarea en las cercanías de una anterior, se tendrán que aplicar restricciones y determinar qué parámetros son importantes.

Una red neuronal profunda consta de múltiples capas de proyección lineal seguidas de elementos no lineales como se ha visto anteriormente en este trabajo. El aprendizaje de una tarea consiste en ajustar el conjunto de pesos (*weights*) y *biases* (*theta*) de las proyecciones lineales para optimizar el rendimiento. Se sabe que se pueden utilizar multitud de configuraciones de *theta* y que muchas de ellas darán como resultado el mismo rendimiento, esto es algo verdaderamente relevante para EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) la sobreparametrización hace que sea bastante probable que haya una solución para una "tarea B" que está cerca de la solución encontrada previamente para la "tarea A".

Mientras se produce el aprendizaje de la "tarea B", EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) protege el desempeño de la "tarea A" restringiendo parámetros con el objetivo de permanecer en una región de bajo error para la "tarea A". Esta restricción está implementada como una penalización cuadrática.

Para justificar la elección de la restricción y definir qué pesos son más importantes para una tarea resulta realmente útil considerar las redes neuronales y su entrenamiento desde un punto de vista probabilístico. Usando este punto de vista, se optimizan los parámetros de forma que se busca los valores más probable dado un conjunto de datos "D". Se puede calcular esta probabilidad condicional $p(\theta|D)$ a partir de la probabilidad previa de los parámetros, $p(\theta)$, y la probabilidad de los datos, $p(D|\theta)$, usando la Regla de Bayes:

$$\log p(\theta|D) = \log p(D|\theta) + \log p(\theta) - \log p(D)$$

Si se asume que el conjunto de datos se divide en dos partes independientes, una para la tarea A (DA) y otra para la tarea B (DB), se puede llevar a cabo una modificación de la anterior fórmula:

$$\log p(\theta|D) = \log p(DB|\theta) + \log p(\theta|DA) - \log p(DB)$$

Por lo tanto, toda la información sobre la tarea A debe haber sido absorbida en la distribución p(θ|DA). Esta probabilidad debe contener información sobre qué parámetros fueron importantes en la tarea A, de esta forma se convierte en la clave de la implementación del algoritmo EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017).

La función de pérdidas que se minimiza en EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) es la siguiente:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

Ilustración 4.5: Fórmula matemática función de pérdidas EWC
*Fuente: **([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017)***

Cuando se pasa a una nueva tarea (por ejemplo, a la tarea C), EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) tratará de mantener los parámetros de la red cercanos a los parámetros aprendidos en la tarea A y en la tarea B.

Esta imagen nos permite comprender el algoritmo de forma visual:

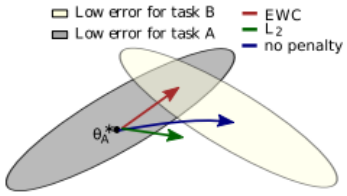

Ilustración 4.6: Esquema funcionamiento algoritmo EWC
*Fuente: **([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017)***

En la imagen se puede apreciar como la flecha roja (EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017)) va desde una región de bajo error de la tarea A hasta una región de bajo error de la tarea B manteniéndose a su vez dentro de la primera región. Esto es lo que diferencia a EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) de otros algoritmos como L2 ([16] Durán, 2019).

Como conclusión se puede decir que este algoritmo busca cambiar los parámetros aprendidos en la tarea A para alcanzar el éxito en la tarea B tratando siempre de mantener el desempeño de la tarea A relativamente constante. De esta forma se lucha para mitigar el gran defecto de las redes neuronales: el olvido catastrófico.

4.2.1.2.2. SI

Se sabe que las redes neuronales biológicas se están adaptando continuamente a cambios de dominios, esto se consigue aprovechando la compleja maquinaria molecular que permite resolver numerosas tareas simultáneamente. A continuación se va a hablar sobre el algoritmo SI ([19] Zenke, Poole, & Ganguli, 2017) (*Synaptic Intelligence*) con el cual se pretende incorporar esta complejidad biológica a las redes neuronales artificiales. Se parte del principio de que cada sinapsis acumula información relevante para la tarea a lo largo del tiempo y explota esta información para almacenar de una forma muy rápida nuevos recuerdos sin olvidar los antiguos. Se ha comprobado que aplicando este algoritmo a tareas de clasificación con aprendizaje continuo se reduca de forma muy significativa el olvido catastrófico a la vez que se mantiene una buena eficiencia a nivel computacional.

Se parte de la base de que en redes neuronales artificiales las sinapsis individuales (pesos) se describen mediante una única cantidad escalar mientras que las sinapsis biológicas individuales hacen uso de maquinaria molecular compleja que puede afectar a la plasticidad a diferentes escalas espacio-temporales.

Las sinapsis que se suelen usar en las redes neuronales artificiales (sinapsis simples) al ser escalares presentan únicamente un estado sináptico de una sola dimensión (1D) sufriendo olvidos catastróficos. Esto se puede solucionar con el uso de sinapsis más complejas que puedan presentar estados sinápticos de tres dimensiones (3D). En el algoritmo SI ([19] Zenke, Poole, & Ganguli, 2017) el estado sináptico rastrea el valor pasado y el actual del parámetro y mantiene una estimación online de la importancia de la sinapsis de cara a resolver problemas encontrados en el pasado. El valor de la importancia se puede calcular de forma eficiente y local en cada sinapsis durante el entrenamiento. Se puede decir que el aprendizaje de tareas futuras está mediado por sinapsis que no eran importantes para tareas anteriores.

Este algoritmo posee un regularizador simple y estructural que puede ser computado de forma online e implementado localmente para cada sinapsis. El objetivo es dotar a cada sinapsis individual de una medida local de importancia en resolver tareas que la red era capaz de resolver en el pasado. Cuando se entrena la red en una nueva tarea se penalizan los cambios en los parámetros importantes con el objetivo de evitar que los viejos recuerdos se sobrescriban. El objetivo final de un entrenamiento exitoso en el cual se utiliza SI ([19] Zenke, Poole, & Ganguli, 2017) consiste en encontrar trayectorias de aprendizaje para las cuales el punto final se encuentra ceca de un mínimo de pérdidas de la función de pérdidas para todas las tareas.

SI en SPLIT MNIST: Para comprender mejor el funcionamiento de este algoritmo de aprendizaje continuado vale la pena conocer su aplicación para “Split MNIST”. Para empezar, se distinguen las siguientes cinco tareas:

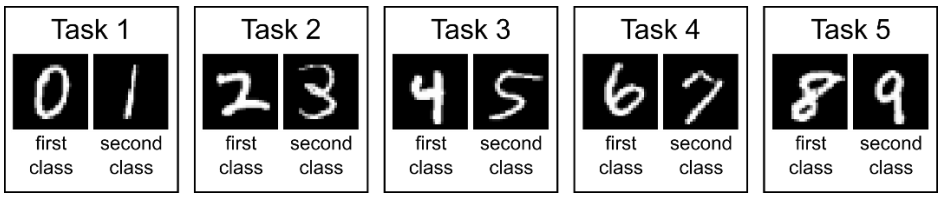

Ilustración 4.7: Ejemplo de split para MNIST
*Fuente: **([35] M. van de Ven, 2019)***

Las cinco tareas consisten en aprender a distinguir entre dos dígitos consecutivos desde 0 hasta 9. Se usa un perceptrón multicapa con sólo dos capas ocultas que consisten en 256 unidades cada una de las cuales con no-linealidades ReLu y con un estándar de *Continual Learning Through Synaptic Intelligence*. Para conseguir una buena actuación con el menor número de épocas se utiliza el optimizador Adam. Este optimizador se resetea después del entrenamiento de cada tarea. Para evaluar el rendimiento se lleva a cabo una comparación entre redes con la dinámica de consolidación sináptica activada (c=1) y desactivada (c=0):

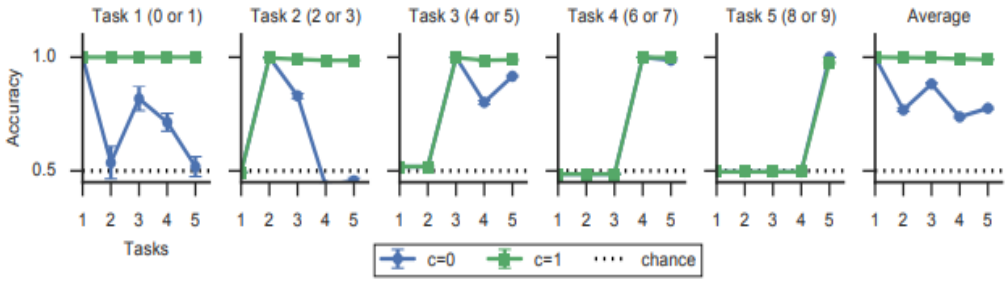

Ilustración 4.8: Gráfico rendimiento con y sin consolidación sináptica
*Fuente: **([19] Zenke, Poole, & Ganguli, 2017)***

Durante el entrenamiento de la primera tarea no la penalización de consolidación es cero porque no hay experiencia anterior con la que las sinapsis puedan regularizar. Cuando se entrena para la segunda tarea, tanto el modelo con consolidación como el modelo sin consolidación muestran una precisión cercana a uno. Sin embargo, sobre la media la red sin consolidación sináptica muestra una considerable pérdida de precisión. En cambio, la red con consolidación sináptica apenas pierde precisión y la media se mantiene cercana a uno en todo momento.

Conclusión: Con la aplicación de este algoritmo se ha demostrado que el problema de los olvidos catastróficos puede ser mitigado permitiendo que las sinapsis individuales estimen su importancia para resolver tareas anteriores o pasadas. De esta forma se sabe que, si se penalizan cambios en las sinapsis más importantes, se pueden aprender tareas nuevas con una interferencia muy pequeña en las tareas aprendidas con anterioridad. Esta penalización de la que se habla es similar a EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017). Sin embargo, esta penalización

en SI ([19] Zenke, Poole, & Ganguli, 2017) se calcula mediante la "fuerza de consolidación por sinapsis" de forma online y sobre la trayectoria entera de aprendizaje mientras que para EWC ([17] Kirkpatricka, Pascanu, Rabinowitz, & al., 2017) la importancia sináptica es calculada offline como la información de Fisher en el mínimo de la pérdida para una determinada tarea.

4.2.1.2.3. MAS

Los humanos pueden aprender de forma continua. El conocimiento antiguo que se utiliza muy poco o nada puede sobrescribirse con nueva información, aunque se suele evitar que se borre un conocimiento adquirido, es muy extraño que el cerebro humano elimine de forma absoluta un conocimiento o habilidad aprendida con anterioridad. En los sistemas de aprendizaje artificial todo se ha centrado en acumular conocimientos y, a su vez, tratar de superar los olvidos catastróficos.

Con MAS ([20] Rahaf Aljundi, 2018) (*Memory Aware Synapses*) los principios mencionados en el párrafo anterior se cuestionan en cierta forma: dada la capacidad limitada que suelen tener todos los modelos y dado que la información que se puede aprender de forma artificial es limitada, el conocimiento debe conservarse o borrarse de manera selectiva.

Para el diseño de MAS ([20] Rahaf Aljundi, 2018) se utilizó como inspiración la neuroplasticidad. Este algoritmo calcula la importancia de los parámetros de una red neuronal de forma no supervisada y online. Dada una nueva muestra con la que se alimenta a la red, lo que MAS ([20] Rahaf Aljundi, 2018) hace es acumular una importancia para cada parámetro, esta importancia está basada en lo sensible que es la función de salida predicha a un cambio en ese determinado parámetro. En el proceso de aprendizaje de una nueva tarea, los cambios en parámetros importantes pueden ser penalizados, esto hace que se pueda prevenir de una forma bastante efectiva que el conocimiento adquirido en tareas anteriores sea sobrescrito. MAS ([20] Rahaf Aljundi, 2018) está basado en el modelo de Hebbian en sistemas biológicos.

La diferencia de MAS ([20] Rahaf Aljundi, 2018) respecto a otros algoritmos de aprendizaje continuado es que puede aprender qué partes del modelo son importantes usando datos sin etiquetar, esto se consigue por la estimación de la importancia de los pesos de los parámetros de la red teniendo en cuenta la sensibilidad de la función de salida. De esta forma MAS ([20] Rahaf Aljundi, 2018) no sólo es capaz de evitar la necesidad de datos etiquetados sino que también evita otras complicaciones dado que consigue la pérdida se encuentre siempre en un mínimo local, lo que hace que los gradientes sean más cercanos a cero. Todo esto hace que MAS ([20] Rahaf Aljundi, 2018) sea un método simple, eficiente y efectivo en aprender qué no hay que olvidar.

Conclusiones: El algoritmo MAS ([20] Rahaf Aljundi, 2018) consiste en estimar la importancia de los pesos para todos los parámetros de la red de una forma no supervisad y online llegando a permitir incluso la adaptación a datos no etiquetados.

- MAS ([20] Rahaf Aljundi, 2018) está inspirado en el esquema de aprendizaje de Hebbian.
- El rendimiento máximo se logra cuando en la salida se utiliza *embedding* en vez de *softmax*.

4.2.2 Pruebas y resultados

A continuación, se lleva a cabo un listado de los resultados obtenidos en este trabajo para el conjunto de datos CONTENEDORES BENIDORM. También se expondrán los resultados obtenidos para el conjunto de datos MNIST. Posteriormente se hará un análisis de todos los resultados y una comparación. Cabe destacar que el objetivo de este trabajo es obtener unos resultados óptimos y razonables para el conjunto de datos CONTENEDORES BENIDORM y para analizar los mismos es necesario establecer una comparación con los resultados obtenidos para MNIST que se sabe que son tanto óptimos como razonables:

Resultados MNIST:

En las siguientes tablas se pueden apreciar los resultados obtenidos para las dos perspectivas globales tomadas en este trabajo en relación al conjunto de datos CONTENEDORES BENIDORM.

La primera perspectiva es la que presenta el siguiente *SPLIT:* {'1': [0, 1], '2': [2, 3], '3': [4, 5], '4': [6, 7]}.

La segunda perspectiva (a la que en este trabajo se la denomina SI vs NO dado que en cada *task* hay una clase de imágenes con contenedores y otra clase imágenes sin contenedores) presenta el siguiente *SPLIT:* {'1': (0, 1), '2': (0, 2), '3': (0, 3), '4': (0, 4), '5': (0, 5), '6': (0, 6), '7': (0, 7)}.

En primer lugar, se exponen los resultados obtenidos para los algoritmos que no emplean Aprendizaje Continuo:

MNIST	
Incremental Task	
Métodos	**Media *(accuracy)***
Adam	94.475673492364921
SGD	97.995674878539596
Adagrad	98.164668439564338
L2	98.274569794466724

Tabla 4.1: Resultados tras la aplicación de los métodos de baselines en el escenario de Incremental Task para MNIST

MNIST	
Incremental Domain	
Métodos	**Media *(accuracy)***
Adam	59.626908628098974
SGD	73.328786888832711
Adagrad	62.734716611062881
L2	69.64767326385119

Tabla 4.2: Resultados tras la aplicación de los métodos de baselines en el escenario de Incremental Domain para MNIST

MNIST	
Incremental Class	
Métodos	**Media *(accuracy)***
Adam	19.694402420574885
SGD	19.018658598083714
Adagrad	19.730511462232602
L2	25.374200226920745

Tabla 4.3: Resultados tras la aplicación de los métodos de baselines en el escenario de Incremental Class para MNIST

A continuación, se exponen las tablas de los resultados obtenidos para el conjunto de datos MNIST

MNIST	
Incremental Task	
Métodos	**Media *(accuracy)***
EWC	97.701568722345132
EWC ONLINE	98.045642312134533
SI	98.562431462356175
MAS	99.602345663326478

Tabla 4.4: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Task para MNIST

MNIST	
Incremental Domain	
Métodos	**Media *(accuracy)***
EWC	61.084348913605616
EWC ONLINE	63.409641482181996
SI	67.10871742693134
MAS	74.83062907572648

Tabla 4.5: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Domain para MNIST

MNIST	
Incremental Class	
Métodos	**Media *(accuracy)***
EWC	19.710539586485122
EWC ONLINE	19.698436712052448
SI	21.123265277517508
MAS	21.561249472479822

Tabla 4.6: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Class para MNIST

Tal y como se puede apreciar en las tablas, los resultados obtenidos para MNIST en el caso de *incremental task* son realmente buenos dado que oscilan entre un 97% y casi un 100% de precisión. Es decir, cumplen de una forma casi perfecta con el objetivo de decir qué digito es dada una tarea conocida, por ejemplo: sabiendo que estamos en la *Task 1* los algoritmos son capaces de decir si la imagen contiene el dígito 0 o del dígito 1. Si se lleva a cabo una comparación con los resultados

obtenidos para otros algoritmos que no emplean Aprendizaje Continuo y que, por lo tanto, tienen elevados costes computacionales y de tiempo, se puede concluir con que el Aprendiza Continuo en el escenario de *incremental task* es óptimo.

Los resultados obtenidos para obtenidos para MNIST en el escenario *incremental domain* puede categorizarse como aceptables, aunque se puede apreciar que la precisión baja bastante para cada algoritmo en comparación al escenario anterior. Es decir, sin una tarea dada o conocida, estos algoritmos son capaces de desempeñar de una forma aceptable su función de concluir si la imagen tratada es de primera clase ([0,2,4,6,8]) o de segunda clase ([1,3,5,7,9]). Si se lleva a cabo una comparación con los métodos que no emplean Aprendizaje Continuo (Adam ([14] Kingma & Lei Ba, 2017), SGD ([13] Roy, 2000), Adagarad ([15] Ruder, 2016), L2 ([16] Durán, 2019)) se puede decir que se llega incluso a mejorar estos métodos dado que el mejor método no continuo es L2 ([16] Durán, 2019) da una precisión en torno a 70 mientras que el mejor método que utiliza Aprendizaje Continuo (MAS ([20] Rahaf Aljundi, 2018)) alcanza una precisión cercana a 75. Por lo tanto, se puede concluir con que los métodos que utilizan Aprendizaje Continuo llegan incluso a superar la precisión de los otros métodos, algo que es realmente interesante dado que la utilización de estos algoritmos trae grandes ventajas consigo.

Los resultados obtenidos para para MNIST en el escenario *incremental class* se pueden considerar no muy óptimos: oscilan entre un 19 y un 22 de precisión aproximadamente. Es decir, los algoritmos o métodos tratados en este trabajo son capaces de lograr una precisión que oscila entre 19 y 22 en el escenario siguiente: sin una tarea dada o conocida deben devolver el dígito que contiene la imagen. Si se lleva a cabo una comparación con los métodos que no utilizan Aprendizaje Continuo se llega a la misma conclusión que en el escenario anterior se obtienen resultados similares en incluso mejores.

Resultados CONTENEDORES BENIDORM:

A continuación, se exponen los resultados para el conjunto de datos CONTENEDORES BENIDORM y sus dos perspectivas expuestas antes:

- Primera perspectiva:

  En primer lugar, se exponen los resultados obtenidos mediante la aplicación de algoritmos baselines que no utilizan Aprendizaje Continuo:

CONTENEDORES BENIDORM	
Incremental Task	
Métodos	**Media *(accuracy)***
Adam	80.96066564407764
SGD	81.028163563453916
Adagrad	82.03456775435671
L2	84.15644567443687

Tabla 4.7: Resultados tras la aplicación de los algoritmos baselines en el escenario de Incremental Task para CONTENEDORES BENIDORM en la primera perspectiva

CONTENEDORES BENIDORM	
Incremental Domain	
Métodos	**Media *(accuracy)***
Adam	56.39226536143875
SGD	59.60796185069589
Adagrad	60.68222773029491
L2	61.05646718901262

Tabla 4.8: Resultados tras la aplicación de los algoritmos baselines en el escenario de Incremental Domain para CONTENEDORES BENIDORM en la primera perspectiva

CONTENEDORES BENIDORM	
Incremental Class	
Métodos	**Media *(accuracy)***
Adam	14.894067796610168
SGD	11.949152542372882
Adagrad	15.145632456785234
L2	17.564335673246467

Tabla 4.9: Resultados tras la aplicación de los algoritmos baselines en el escenario de Incremental Domain para CONTENEDORES BENIDORM en la primera perspectiva

En segundo lugar, se exponen los resultados obtenidos tras la aplicación de los algoritmos que utilizan Aprendizaje Continuo:

CONTENEDORES BENIDORM	
Incremental Task	
Métodos	**Media *(accuracy)***
EWC	84.66176256975083
EWC ONLINE	81.16700873601397
SI	82.84303045440855
MAS	90.06558896933996

Tabla 4.10: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Task para CONTENEDORES BENIDORM en la primera perspectiva

CONTENEDORES BENIDORM	
Incremental Domain	
Métodos	**Media *(accuracy)***
EWC	57.97743965694876
EWC ONLINE	62.61359128104298
SI	84.60618255654371
MAS	77.62456788654345

Tabla 4.11: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Domain para CONTENEDORES BENIDORM en la primera perspectiva

CONTENEDORES BENIDORM	
Incremental Class	
Métodos	**Media *(accuracy)***
EWC	15.10593220338983
EWC ONLINE	17.012711864406782
SI	17.594984681431093
MAS	23.50063535436531

Tabla 4.12: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Class para CONTENEDORES BENIDORM en la primera perspectiva

A continuación, se va a proceder a analizar los resultados obtenidos para la primera perspectiva del conjunto de datos CONTENEDORES BENIDORM. Se parte de la base de que el gran éxito de este trabajo reside en haber logrado adaptar las imágenes de este conjunto de datos proporcionado por el VPU de la EPS (UAM) de forma que se hayan podido ejecutar los algoritmos de Aprendizaje Continuo y se pueda llevar a cabo esta evaluación de resultados. Para lograrlo fue necesario un complicado preprocesamiento (detallado en la sección anterior de PREPROCESAMIENTO). Una vez aclarado esto se puede proceder a la interpretación general de los resultados.

Para el escenario de *incremental task* se obtienen unos resultados muy óptimos. Cabe destacar el gran resultado de precisión obtenido por el método MAS ([20] Rahaf Aljundi, 2018), que en todas las pruebas ha oscilado en torno al 90%. Es decir, los métodos de Aprendizaje Continuo tratados en este trabajo desempeñan de una forma realmente óptima la función de, con una tarea/*task* dada o conocida, devolver qué tipo de contenedor aparece en la imagen (por ejemplo, para la *task 2* devuelven de una forma realmente óptima si el contenedor de la imagen es de tipo 2 o de tipo 3). Si se lleva a cabo una comparación con los resultados obtenidos para MNIST se puede decir que los resultados son ligeramente peores dado que para el conjunto de datos MNIST los resultados en general de estos métodos suelen situarse por encima de 95 de precisión. Sin embargo, y teniendo en cuenta la dificultad de disitinguir los distintos tipos de contenedor y teniendo en cuenta que en el conjunto de datos CONTENEDORES BENIDORM hay muchas más imágenes sin contenedor que con contenedor, se puede decir que los resultados obtenidos en el escenario de *incremental task* son verdaderamente óptimos.

Para el escenario de *incremental domain* se obtienen unos resultados aceptables. Algunos métodos como SI ([19] Zenke, Poole, & Ganguli, 2017) o MAS ([20] Rahaf Aljundi, 2018) logran resultados que se pueden valorar como verdaderamente positivos. Es decir, estos métodos obtienen buenos resultados en el desempeño de la función de, sin una tarea dada o conocida, devolver si la imagen pertenece a los tipos [0, 2, 4, 6] o a los tipos [1, 3, 5, 7]. Si se comparan estos resultados con los obtenidos para MNIST en el escenario *incremental domain* se puede decir que son bastante similares y que incluso algún método como MAS ([20] Rahaf Aljundi, 2018) en CONTENEDORES BENIDORM mejora de forma considerable el mejor resultado obtenido para MNIST. Esto puede deberse de nuevo a que una clase de las imágenes no contenga contenedores y suponga un elevado porcentaje de las imágenes del conjunto de datos CONTENEDORES BENIDORM.

Para el escenario de *incremental class* se obtienen de nuevo resultados bastante bajos dada la dificultad que supone desempeñar la tarea pretendida en este escenario: sin una tarea dada o conocida, los algoritmos deben devolver exactamente el tipo de contenedor que aparece en la imagen. Cabe destacar de nuevo el resultado obtenido por el método MAS ([20] Rahaf Aljundi, 2018) dado que es ligeramente mejor que el resto de algoritmos. En comparación con los resultados

obtenidos para MNIST se puede apreciar una ligera mejora pero en cierta forma irrelevante dado que cada vez que se ejecutan los métodos varían un poco sus resultados por lo que una diferencia tan pequeña no es realmente significativa. Aun así, el simple hecho de obtener resultados similares a los obtenidos para MNIST ya es un éxito dado que el objetivo principal de este trabajo es probar estos algoritmos o métodos de aprendizaje continuo en un conjunto de datos compuesto por imágenes "normales" demostrando así que se pueden utilizar estos algoritmos de forma cotidiana dentro del mundo del *Deep Learning*.

- Segunda perspectiva:

CONTENEDORES BENIDORM si vs no	
Incremental Task	
Métodos	**Media *(accuracy)***
EWC	98.63456722345556
EWC ONLINE	97.73568854367123
SI	99.44566334546643
MAS	99.56782346773323

Tabla 4.13: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Task para CONTENEDORES BENIDORM en la segunda perspectiva

CONTENEDORES BENIDORM si vs no	
Incremental Domain	
Métodos	**Media (accuracy)**
EWC	98.55329034333514
EWC ONLINE	97.69575577821324
SI	99.14306167008566
MAS	99.15567445663231

Tabla 4.14: Resultados tras la aplicación de los métodos de Aprendizaje Continuo en el escenario de Incremental Domain para CONTENEDORES BENIDORM en la segunda perspectiva

CONTENEDORES BENIDORM si vs no	
Incremental Class	
Métodos	**Media *(accuracy)***
EWC	97.56784324567323
EWC ONLINE	97.42406438458369
SI	99.11234454256479
MAS	99.13012345322234

Tabla 4.15: Resultados tras la aplicación de ls métodos de Aprendizaje Continuo en el escenario de Incremental Class para CONTENEDORES BENIDORM en la segunda perspectiva

El análisis de estos resultados es menos complejo y tedioso que los anteriores dado que en líneas generales los resultados oscilan entre en un 96 y un 99 de precisión. Estos resultados tan elevados y, a su vez, tan buenos se deben a que en cada *task* se "enfrenta" una clase que contiene un tipo de contenedor y la clase que no contiene contenedor. Dado que el principal comportamiento de estos algoritmos es utilizar/aprovechar/no desperdiciar el conocimiento obtenido (cada algoritmo o método lo hace de una forma diferente) en tareas o situaciones anteriores pues tiene bastante sentido que los resultados sean tan elevados dado que siempre se hace "lo mismo" por lo que el conocimiento adquirido con anterioridad es realmente valioso para las siguientes tareas o situaciones.

4.2.3 Pruebas visuales

En esta sección se va a exponer de forma visual diferentes aspectos explicados anteriormente con el objetivo de facilitar la comprensión de los mismos.

En primer lugar, se va a demostrar de forma visual el resultado final del pre-procesamiento explicado en la sección anterior dedicada a este tema. Como se ha resaltado anteriormente en este trabajo, gran parte del éxito conseguido en la realización de este proyecto ha residido en el pre-procesamiento:

- Imagen original de un contenedor del conjunto de datos CONTENEDORES BENIDORM:

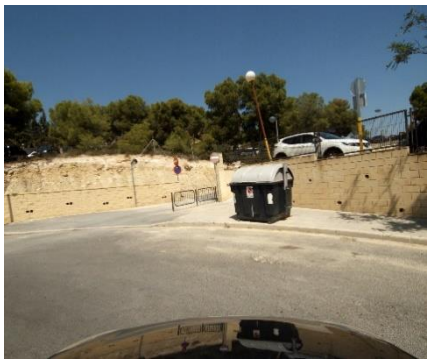

Ilustración 4.9: Imagen original conjunto de datos CONTENEDORES BENIDORM

- Prueba visual de la primera fase del pre-procesamiento que consiste en recortar el contenedor de la imagen:

Ilustración 4.10: Resultado primera fase pre-procesamiento

- Prueba visual de la última fase del pre-procesamiento, en la cual se dejan las imágenes listas para ser introducidas en los métodos tratados en este trabajo:

Ilustración 4.11: Resultado última fase pre-procesamiento

Por último, se van a mostrar algunas pruebas visuales de lo que serían algunas *tasks* integradas por dos clases: la primera clase contiene una imagen sin contenedor y la segunda clase contiene una imagen con un tipo de contenedor. El funcionamiento de todos los métodos tratados en este trabajo está basado en estas *tasks.*

- *Task 1: no contenedor vs contenedor tipo 1:*

 

Ilustración 4.12: Ejemplo gráfico Task 1

- *Task 2: no contenedor vs contenedor tipo 2:*

 

Ilustración 4.13: Ejemplo gráfico Task 2

- *Task 3: no contenedor vs contenedor tipo 3:*

 

Ilustración 4.14: Ejemplo gráfico Task 3

- *Task 4: no contenedor vs contenedor tipo 4:*

 

Ilustración 4.15: Ejemplo gráfico Task 4

5. Conclusiones y trabajo futuro

5.1. Conclusiones

Este trabajo puede tomarse como un punto de partida en la consecución de un cambio totalmente necesario en el mundo del *Deep Learning*. Con este trabajo se demuestra la posibilidad de aplicar métodos o algoritmos basados en el Aprendizaje Continuo a conjuntos de datos estándar como es el caso del *dataset* CONTENEDORES BENIDORM. Tras evaluar los resultados obtenidos tanto para MNIST como para CONTENEDORES BENIDORM se puede concluir con que los algoritmos o métodos basados en Aprendizaje Continuo llegan a igualar e incluso a mejorar los resultados de otro tipo de métodos o algoritmos utilizados de forma frecuente en *Deep Learning*.

El éxito de este trabajo reside en los resultados obtenidos para CONTENEDORES BENIDORM dado que el arduo pre-procesamiento ha permitido que los algoritmos de Aprendizaje Continuo den resultados muy aceptables e interesantes en comparación con otro tipo de algoritmos y con los resultados obtenidos para el conjunto de datos MNIST, que no deja de ser un *dataset* creado y diseñado para llevar a cabo investigaciones y diferentes tipos de pruebas.

El Aprendizaje Continuo es algo que debe terminar implantándose dentro del *Deep Learning* dado que el famoso problema de los olvidos catastróficos es una traba bastante considerable dados los problemas que acarrea: gasto computacional, largos tiempos de espera, gasto de almacenamiento…

5.2. Trabajo futuro

Teniendo en cuenta los resultados obtenidos en este trabajo, se proponen las siguientes ideas para, en un futuro, proseguir con la tarea de implantación del Aprendizaje Continuo de forma definitiva en el *Deep Learning*:

- Buscar nuevas formas de implantar el Aprendizaje Continuo en el escenario de *incremental class* para tratar de aumentar la precisión obtenida en los resultados de este trabajo, ya que es considerablemente baja.
- Investigar nuevos algoritmos basados en Aprendizaje Continuo dado que en la actualidad el número de algoritmos diseñados es bastante bajo.
- Incrementar el número de investigaciones sobre estos algoritmos en conjuntos de datos estándar similares al utilizado en este trabajo, ya que son los *datasets* que se suelen utilizar de forma normal.

Bibliografía

[1] Yen-Chang Hsu, Y.-C. L. (2018). *"Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines"*. Obtenido de https://arxiv.org/abs/1810.12488

[2] Lancomilla, P. (2016). *https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.pd*. Obtenido de "Deep Learning: Redes".

[3] Bagnato, J. I. (298 de noviembre de 2018). *blog Aprende Machine Learnin*. Obtenido de ¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[4] Chollet, F. (2017). Deep Learning with Python. En F. Chollet, *Deep Learning with Pytho* (pág. 384).

[5] LeCun Yann, C. C. (s.f.). *"The MNIST Database of handwritten digits"*. Obtenido de http://yann.lecun.com/exdb/mnist/

[6] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. Obtenido de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf

[7] Tran, K. (Abril de 2020). *What is PyTorch?* Obtenido de https://towardsdatascience.com/what-is-pytorch-a84e4559f0e3

[8] Dipanjan (DJ), S. (2018). *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*. Obtenido de https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[9] Brownlee, J. P. (diciembre de 2017). *A Gentle Introduction to Transfer Learning for Deep Learning*. Obtenido de https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[10] Aljundi, R. (septiembre de 2019). *Continual Learning in Neural*. Obtenido de https://arxiv.org/pdf/1910.02718.pdf

[11] Maltoni, D., & Lomonaco, V. (junio de 2018). *Continuous Learning in Single-Incremental-Task Scenarios*. Obtenido de https://www.researchgate.net/publication/325965452_Continuous_Learning_in_Single-Incremental-Task_Scenarios

[12] Gido M. van de Ven1, 2. A. (abril de 2019). *"Three scenarios for continual learning"*. Obtenido de https://arxiv.org/pdf/1904.07734.pdf

[13] Roy, R. (mayo de 2000). *ML | Stochastic Gradient Descent (SGD)*. Obtenido de https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[14] Kingma, D. P., & Lei Ba, J. (enero de 2017). *"ADAM: a method for stochastic optimization"*. Obtenido de https://arxiv.org/pdf/1412.6980.pdf

[15] Ruder, S. (enero de 2016). *"An overview of gradient descent optimization algorithms"*. Obtenido de https://ruder.io/optimizing-gradient-descent/

[16] Durán, J. (octubre de 2019). *"Técnicas de Regularización Básicas para Redes Neuronales"*. Obtenido de https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4

[17] Kirkpatricka, J., Pascanu, R., Rabinowitz, N., & al., e. (enero de 2017). *"Overcoming catastrophic forgetting in neural"*. Obtenido de https://arxiv.org/pdf/1612.00796.pdf

[18] Schwarz, J., Luketina, J., & Czarnecki, W. M. (julio de 2018). Obtenido de "Progress & Compress: A scalable framework for continual learning": https://arxiv.org/pdf/1805.06370.pdf

[19] Zenke, F., Poole, B., & Ganguli, S. (junio de 2017). Obtenido de "Continual Learning Through Synaptic Intelligence": https://arxiv.org/pdf/1703.04200.pdf

[20] Rahaf Aljundi, F. B. (2018). *Memory Aware Synapses: Learning what (no)t to forget*. Obtenido de https://openaccess.thecvf.com/content_ECCV_2018/papers/Rahaf_Aljundi_Memory_Aware_Synapses_ECCV_2018_paper.pdf

[21] Caparrini, F. S. (2019). *Redes Neuronales: una visión superificial*. Obtenido de http://www.cs.us.es/~fsancho/?e=72

[22] Cancela, J. (2017). *El Perceptrón*. Obtenido de https://www.javiercancela.com/2017/03/19/python-machine-learning-ii/

[23] Kane, M. (2017). *Bringing Machine Learning to your iOs apps*. Obtenido de https://academy.realm.io/posts/altconf-2017-meghan-kane-bringing-machine-learning-to-your-ios-apps/

[24] Pascual, C. (2018). *Understanding Regression Error Metrics in Python*. Obtenido de https://www.dataquest.io/blog/understanding-regression-error-metrics/

[25] Developers Google. (s.f.). *Aprendizaje Automático*. Obtenido de https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?hl=es-419

[26] Gondhalekar, A. (2020). *Data Augmentation – Is it really necessary?* Obtenido de https://medium.com/analytics-vidhya/data-augmentation-is-it-really-necessary-b3cb12ab3c3f

[27] López González, R. (2017). *Detección Automática de Órganos en adquisiciones de Tomografía Computarizada con Métodos de Machine Learning*. Obtenido de https://riunet.upv.es/bitstream/handle/10251/90078/L%C3%93PEZ%20-%20Detecci%C3%B3n%20autom%C3%A1tica%20de%20%C3%B3rganos%20en%20adquisiciones%20de%20TC%20mediante%20m%C3%A9todos%20de%20aprendizaje%20au....pdf?sequence=1

[28] Bagnato, J. (2018). *¿Cómo funcionan las Convolutionan Neural Networks?* Obtenido de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[29] Muhamad, Y. e. (Muhamad Yani et al 2019 J. Phys.: Conf. Ser. 1201 012052 de 2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. *Journal of Physics: Conference Series*, 1201. Obtenido de https://iopscience.iop.org/article/10.1088/1742-6596/1201/1/012052/pdf

[30] Numerntur.org. (s.f.). Funciones de Activación en Softmax. http://numerentur.org/funcion-de-activacion-softmax/ Funciones de Activación en Softmax.

[31] Lim, S.-H. &. (2016). *An analysis of image storage systems for scalable training of deep neural networks*. Obtenido de https://www.researchgate.net/publication/306056875_An_analysis_of_image_storage_systems_for_scalable_training_of_deep_neural_networks

[32] Hussain, M. (2020). *Introduction to Transfer Learning | What is Transfer Learning in Deep Learning*. Obtenido de https://www.mygreatlearning.com/blog/transfer-learning/

[33] Lomonaco, V. (2020). *Continual Learning in Comptuer Vision Competition: Approaches, Results, Current Challenges and Future Directions*. Obtenido de https://www.catalyzex.com/author/Vincenzo%20Lomonaco

[34] Hyungeun, C., Seunghyoung, R., & Hongseok, K. (2018). *Short-Term Load Forecasting based on ResNet and LSTM*. Obtenido de https://www.researchgate.net/figure/The-structure-of-ResNet-12_fig1_329954455

[35] M. van de Ven, G. (2019). *Three scenarios for Continual Learning*. Obtenido de https://deepai.org/publication/three-scenarios-for-continual-learning

[36] Rahul, R. (2020). *Stochastic Gradient Descent (SGD)*. Obtenido de https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[37] An, Z., Yu, M., & Changjian, Z. (s.f.). *An improved Adam Algorithm using look-ahead 2017*. Obtenido de https://www.researchgate.net/figure/Comparison-between-Adam-and-AWL-on-Logistic-Regression-Logistic-regression-validation_fig1_317920873